

Caminhos mínimos

1. Grafos com pesos nas arestas: $G = (V, E)$, $c: E \rightarrow \mathbb{R}$ (usualmente, $c(e) \geq 0$ para toda $e \in E$, mas em alguns casos consideraremos comprimentos negativos).

Caminhos mínimos

1. Grafos com pesos nas arestas: $G = (V, E)$, $c: E \rightarrow \mathbb{R}$ (usualmente, $c(e) \geq 0$ para toda $e \in E$, mas em alguns casos consideraremos comprimentos negativos). **Redes** = Networks
2. Queremos: caminhos de peso/comprimento mínimo em G

Variantes do problema

Variantes do problema

- ▶ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).

Variantes do problema

- ▷ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).
- ▷ Dado s queremos um caminho de comprimento mínimo de s a t , para todo $t \in V(G)$ (caminhos mínimos de uma fonte).

Variantes do problema

- ▷ Dados s e t , queremos um caminho mínimo de s a t em G (problema fonte-sorvedouro).
- ▷ Dado s queremos um caminho de comprimento mínimo de s a t , para todo $t \in V(G)$ (caminhos mínimos de uma fonte).
- ▷ Queremos um caminho de comprimento mínimo de s a t , para todo s e para todo $t \in V(G)$ (caminhos mínimos entre todos os pares).

Princípio básico: relaxação

Princípio básico: relaxação

- ▶ Relaxação em arestas

Princípio básico: relaxação

- ▷ Relaxação em arestas
- ▷ Relaxação em vértices

Relaxação em arestas (caminhos mínimos a partir de uma fonte)

Relaxação em arestas (caminhos mínimos a partir de uma fonte)

Suponha que temos um arco $e=(v, w)$ com cauda v e cabeça w em G .

Relaxação em arestas (caminhos mínimos a partir de uma fonte)

Suponha que temos um arco $e=(v, w)$ com cauda v e cabeça w em G . Ao considerarmos o arco e , temos um caminho mais curto de s a w ?

Relaxação em arestas (caminhos mínimos a partir de uma fonte)

Suponha que temos um arco $e=(v,w)$ com cauda v e cabeça w em G . Ao considerarmos o arco e , temos um caminho mais curto de s a w ?

```
if (wt[w] > wt[v] + e.wt)
  { wt[w] = wt[v] + e.wt; st[w] = v; }
```

Relaxação em arestas (caminhos mínimos a partir de uma fonte)

Suponha que temos um arco $e=(v, w)$ com cauda v e cabeça w em G . Ao considerarmos o arco e , temos um caminho mais curto de s a w ?

```
if (wt[w] > wt[v] + e.wt)
  { wt[w] = wt[v] + e.wt; st[w] = v; }
```

▷ O vetor $st[]$ define uma *árvore de de caminhos mínimos* enraizada em s .

Relaxação em vértices (caminhos mínimos entre todos os pares)

Relaxação em vértices (caminhos mínimos entre todos os pares)

É melhor passar por x para ir de s a t ?

Relaxação em vértices (caminhos mínimos entre todos os pares)

É melhor passar por x para ir de s a t ?

```
if (d[s][t] > d[s][x] + d[x][t])  
  { d[s][t] = d[s][x] + d[x][t]; p[s][t] = p[s][x]; }
```

Relaxação em vértices (caminhos mínimos entre todos os pares)

É melhor passar por x para ir de s a t ?

```
if (d[s][t] > d[s][x] + d[x][t])  
  { d[s][t] = d[s][x] + d[x][t]; p[s][t] = p[s][x]; }
```

(Vale a pena lembrar: algoritmo de Warshall para o fecho transitivo)

Algoritmo de Dijkstra: caminhos mínimos a partir de uma fonte s

Algoritmo de Dijkstra: caminhos mínimos a partir de uma fonte s

- ▶ Fazemos uma busca generalizada a partir de s , usando como fila generalizada uma fila de prioridade:

Algoritmo de Dijkstra: caminhos mínimos a partir de uma fonte s

- ▶ Fazemos uma busca generalizada a partir de s , usando como fila generalizada uma fila de prioridade: o vértice de maior prioridade na fronteira (*fringe*) é aquele de ‘menor distância’!

Algoritmo de Dijkstra: caminhos mínimos a partir de uma fonte s

- ▶ Fazemos uma busca generalizada a partir de s , usando como fila generalizada uma fila de prioridade: o vértice de maior prioridade na fronteira (*fringe*) é aquele de ‘menor distância’!

???:

Algoritmo de Dijkstra: caminhos mínimos a partir de uma fonte s

- ▷ Fazemos uma busca generalizada a partir de s , usando como fila generalizada uma fila de prioridade: o vértice de maior prioridade na fronteira (*fringe*) é aquele de ‘menor distância’!

???: Como determinar a prioridade dos vértices?

- ▷ **Restrição importante:** supomos que não há arcos de custos negativos no grafo.

Algoritmo de Dijkstra: correção

Algoritmo de Dijkstra: correção

- ▶ Construimos uma árvore, mantemos os vértices já vistos na fronteira. Mantemos os vetores $wt []$ e $st []$.

Algoritmo de Dijkstra: correção

- ▷ Construimos uma árvore, mantemos os vértices já vistos na fronteira. Mantemos os vetores $wt[]$ e $st[]$.
- ▷ Invariante:
 - (a) A árvore T dada por $st[]$ é uma árvore de caminhos mínimos, e $wt[v]$ é a distância de s a v para todo vértice em T .
 - (b) Se w está na fronteira, então $wt[w]$ é
$$\min\{c(P) : P \text{ é um } s-w \text{ caminho com } P - w \text{ contido em } T\};$$
em palavras: $wt[w]$ é o menor comprimento de um caminho de w a s que está contido em T , a menos de w e da aresta incidente a w . Ademais, o arco de um tal caminho mínimo P incidente a w tem extremos w e $st[w]$.

Programa 21.1, Algoritmo de Dijkstra

Programa 21.1, Algoritmo de Dijkstra

```
#define GRAPHpfs GRAPHspt
#define P (wt[v] + t->wt)
void GRAPHpfs(Graph G, int s, int st[], double wt[])
{ int v, w; link t;
  PQinit(); priority = wt;
  for (v = 0; v < G->V; v++)
    { st[v] = -1; wt[v] = maxWT; PQinsert(v); }
  wt[s] = 0.0; PQdec(s);
  while (!PQempty())
    if (wt[v = PQdelmin()] != maxWT)
      for (t = G->adj[v]; t != NULL; t = t->next)
        if (P < wt[w = t->v])
          { wt[w] = P; PQdec(w); st[w] = v; }
}
```

Programa 21.1, Algoritmo de Dijkstra (cont.)

Programa 21.1, Algoritmo de Dijkstra (cont.)

- ▶ Basicamente igual ao Programa 20.4, mas com `#define P t->wt` trocado por `#define P (wt[v] + t->wt)`

Programa 21.1, Algoritmo de Dijkstra (cont.)

- ▶ Basicamente igual ao Programa 20.4, mas com `#define P t->wt` trocado por `#define P (wt[v] + t->wt)` (qual é a outra diferença entre os Programas 20.4 e 21.1?)

Programa 21.1, Algoritmo de Dijkstra (cont.)

- ▶ Basicamente igual ao Programa 20.4, mas com `#define P t->wt` trocado por `#define P (wt[v] + t->wt)` (qual é a outra diferença entre os Programas 20.4 e 21.1?)
- ▶ Fazemos no máximo n inserções, n remoções de mínimo, e m reduções de prioridade.

Programa 21.1, Algoritmo de Dijkstra (cont.)

- ▶ Basicamente igual ao Programa 20.4, mas com `#define P t->wt` trocado por `#define P (wt[v] + t->wt)` (qual é a outra diferença entre os Programas 20.4 e 21.1?)
- ▶ Fazemos no máximo n inserções, n remoções de mínimo, e m reduções de prioridade.
- ▶ Usando um *heap* binário, Dijkstra tem complexidade $O(m \log n)$.